Algorithms and Software for PDE's with AMR

Dan Martin
Applied Numerical Algorithms Group
Lawrence Berkeley Laboratory

April 23, 2002

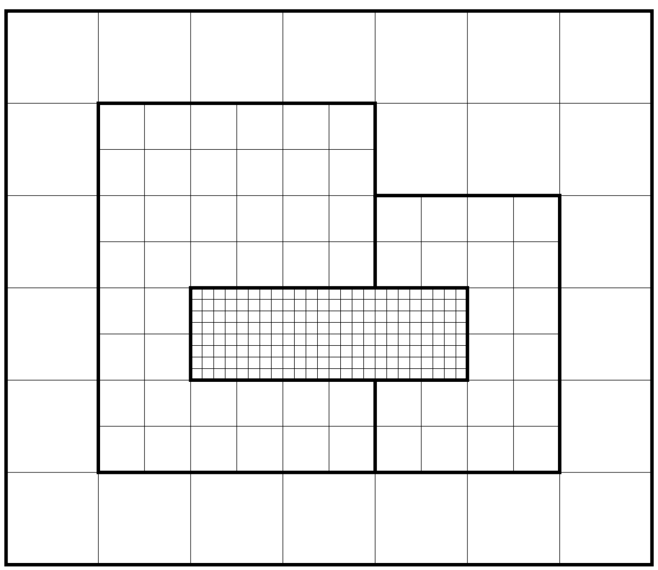
Outline

1. AMR Overview
2. Algorithm refinements for AMR
3. Chombo
4. Particles and Chombo
5. Examples

Adaptive Mesh Refinement (AMR)
(Berger & Oliger, 1984):

Approach:

- locally refine patches of the domain where needed to improve solution
- each patch is a logically rectangular structured grid
  - better efficiency of data access
  - can amortize overhead of irregular operations over large number of regular operations
- refined grids are dynamically created and destroyed



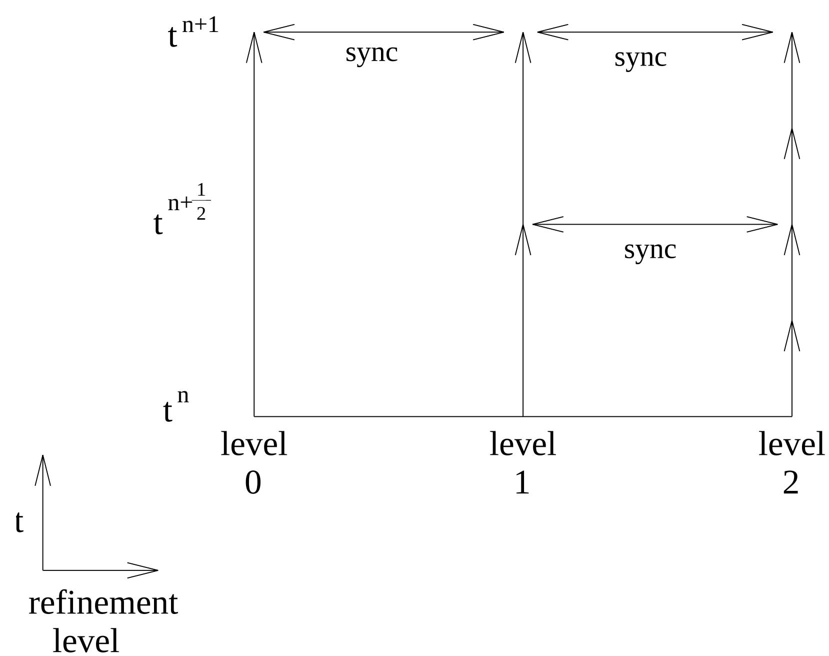
Refine in time as well as space (subcycling)

Advantages:

- better efficiency (CFL condition)
- everything at same CFL number can improve advection performance

Disadvantage:

- Cause of much of the work!



Algorithm refinements for AMR

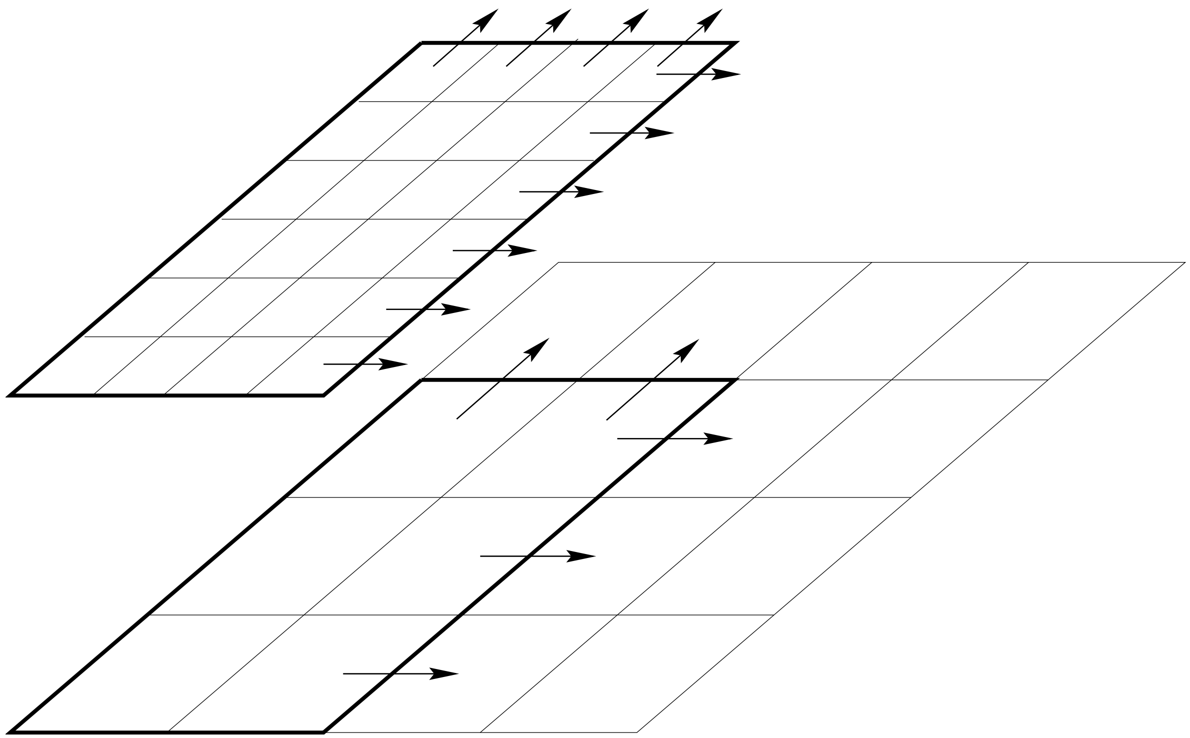
- Discontinuities in Grid Spacing
- Coupling coarse and fine solutions appropriately
- Regridding/reinitialization

Coupling coarse and fine solutions

- Fine grids – need boundary conditions from coarse grids – interpolated in time and space.
- Coarse grid – may need to see effect of fine-grid solution
  example: flux correction for conservation.
- maintain constraints in the presence of AMR
  - conservation – flux correction due to mismatch
  - divergence constraints (incompressible flow, MHD)
  - freestream preservation – incompressible flow; scalar initialized to a constant should remain constant.

Conservation:

To maintain conservation, the difference between the coarse grid flux and the sum of the fine grid fluxes is “refluxed” into the coarse cells adjacent to the fine grids.

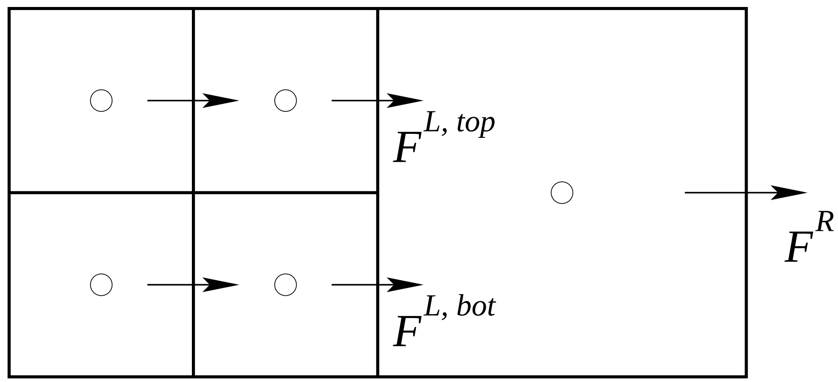


Discontinuities in Grid Spacing

- In a single-grid world, rely on grid regularity for accuracy – cancellations, etc.
- Local refinement breaks grid regularity – loss of accuracy can result
- Lack of smoothness at grid interfaces is another common problem.
- If not careful, can lose the accuracy benefit of local refinement from additional errors induced at coarse-fine interfaces

- Example – Poisson's Equation ($\Delta\phi = \rho$)
  - "Elliptic Matching condition" – need both ϕ and $\frac{\partial\phi}{\partial n}$ to be continuous at the coarse-fine interface. Otherwise, charge induced at interface.
  - Accuracy – because interpolated value is divided by h^2, need at least quadratic interpolation for fine-grid boundary conditions.
  - Solution is to define composite operators which satisfy both of these constraints by using flux-matching and quadratic interpolation. Result: closer coupling of coarse- and fine- solutions.

Truncation Errors at Coarse-fine interfaces



$$
\begin{aligned}
D^*F &= \frac{1}{\Delta x}(F^R - \frac{1}{2}(F^{L,top} + F^{L,bot})) \\
&= \frac{1}{\Delta x}(F((i+\frac{1}{2})\Delta x, \overline{y}) - (F((i-\frac{1}{2})\Delta x, \overline{y}) + C(\Delta x)^2)) \\
&= \frac{\partial F}{\partial x} + O(\Delta x) \quad (\textbf{not } O(\Delta x^2))
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial U^{mod}}{\partial t} + \nabla \cdot F^{mod} \ = \ \tau \quad &= \quad O(\Delta x) \ at \ C/F \ boundary \\
&= \quad O(\Delta x^2) \ elsewhere
\end{aligned}
$$

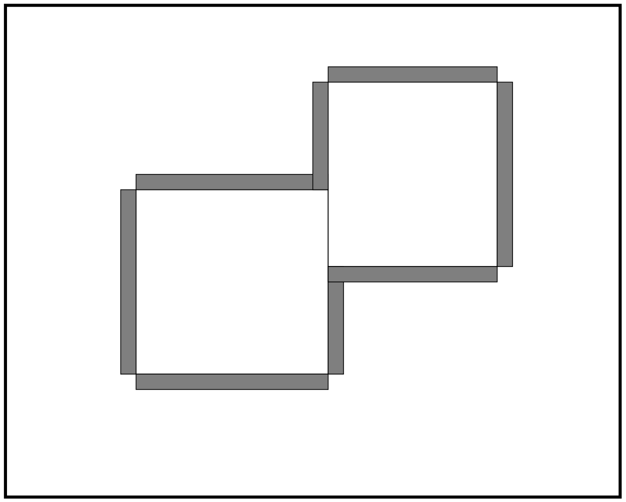
Discretizing Elliptic PDE's

Naive approach:

- Solve $\Delta\psi^c = g^c$ on coarse grid.

- Solve $\Delta\psi^f = g^f$ on fine grid, using coarse grid values to interpolate boundary conditions.

Such an algorithm yields coarse-grid solution accuracy on the fine grid (Bai and Brandt, Thompson and Ferziger).



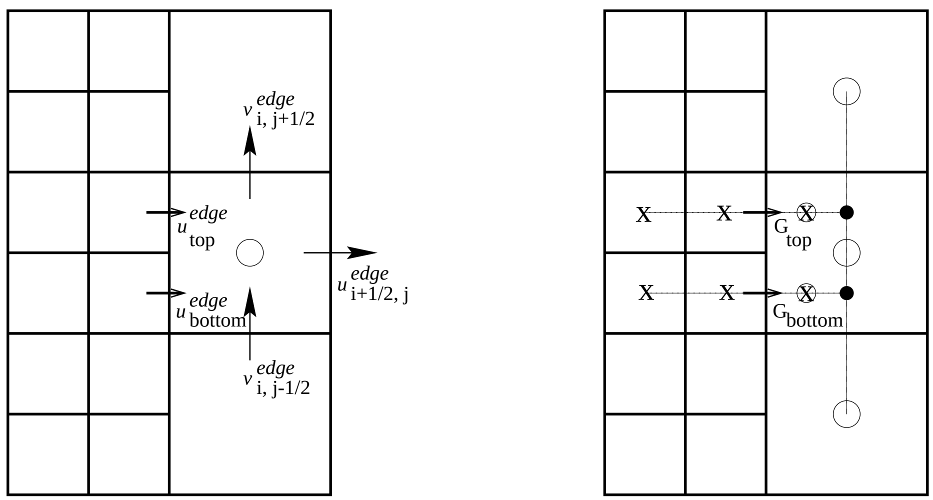
$\psi^c \approx \Delta^{-1}(g + \tau^c)$. Using ψ^c to interpolate boundary conditions for fine calculation introduces coarse-grid error on fine grid.

Solution: compute ψ^{comp}, the solution of the properly-posed problem on the composite grid.

$$\begin{aligned} \Delta^c \psi^{comp} &= g^c \text{ on } \Omega^c - \mathcal{C}(\Omega^f) \\ \Delta^f \psi^{comp} &= g^f \text{ on } \Omega^f \end{aligned}$$

$$[\psi] = 0, \quad \left[\frac{\partial \psi}{\partial n}\right] = 0 \text{ on } \partial\Omega^{c/f}$$

The Neumann matching conditions are flux matching conditions, and are discretized by computing a single-valued flux at the boundary.



Modified equation: $\psi^{comp} = \psi + \Delta^{-1}\tau^{comp}$, where τ is a local function of the solution derivatives.

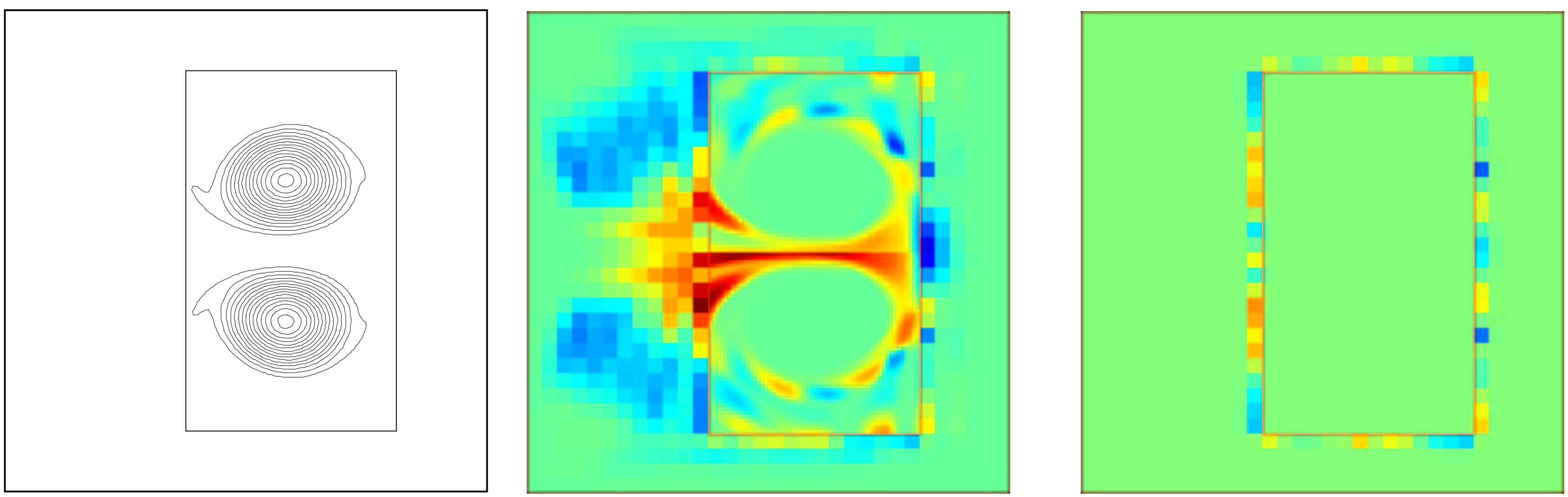
Divergence constraint

- incompressible flow: $\nabla \cdot \vec{u} = 0$.
- Use projection method to ensure that divergence constraint is met.
- Because AMR timestep does subcycled level-by-level advances, resulting solution will not satisfy divergence constraint across the entire hierarchy of refined levels.
- Solution – apply projection based on composite operators during synchronization step to ensure constraint is met.

Freestream Preservation

- For incompressible Navier-Stokes code, compute incompressible advection velocities with which to compute advection.
- Since these velocities are computed on a level-by-level basis, not divergence-free in a composite sense; refluxing for conservation results in a constant field losing its const-ness
- Define auxiliary advected scalar set to $1 \rightarrow \Lambda \neq 1$ is a measure of failure, which can be used to compute a correction, applied to subsequent advection velocities (lagged correction).

$$\nabla \cdot \vec{u} = \eta(\Lambda - 1), \quad \eta = O(\frac{1}{\Delta t})$$

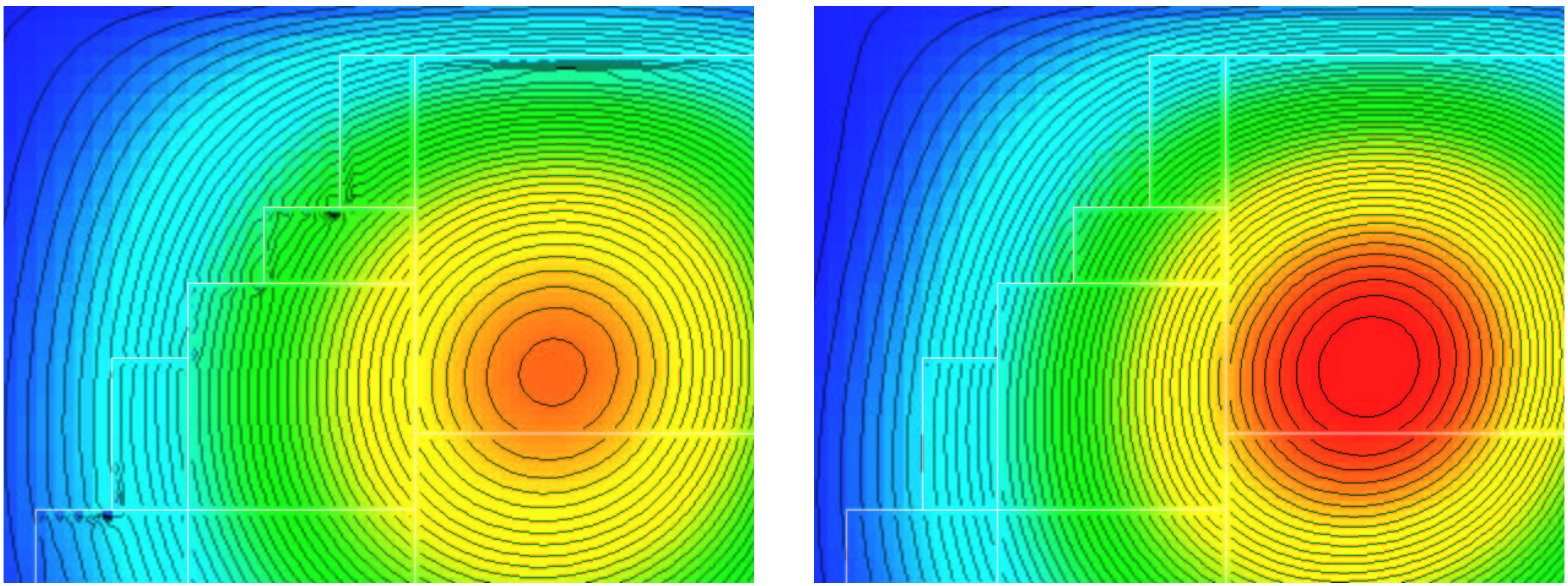


Algorithm Adjustments for AMR

- explicit algorithms generally "easier" than implicit
  - implicit normally requires elliptic solves at synchronization
  - coarse-fine boundary conditions not always obvious
- casting in terms of fluxes at faces simplifies matching conditions at coarse-fine interfaces
- Crank-Nicolson (everybody's favorite 2nd-order semi-implicit method) can be problematic for AMR.
  - Presence of sharp source terms can cause C-N problems (not L_0 stable).
  - Can swich to backward Euler (L_0 stable, but 1^{st} order in time).
  - Second-order Runge-Kutta method (Twizell, Gumel, Arigu, 1996), gives 2nd-order in time, L_0 stability, but costs an additional elliptic solve.

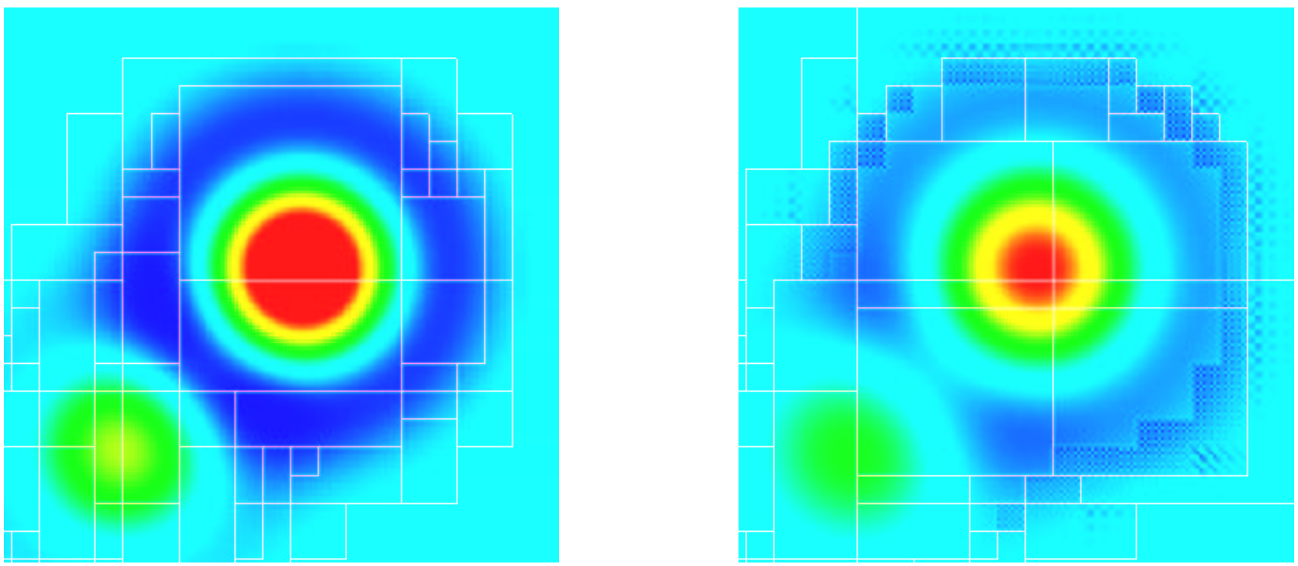
Time-dependent Ginsberg-Landau Equation results

Laplacian(ϕ) with Crank-Nicolson vs. Backwards Euler



Regridding/reinitialization

- Want to regrid often, to follow changing solution, minimize “buffer” refined cells, **but...**
- Interpolate new fine-level data from coarse-level data – if conservative, not necessarily accurate or smooth enough
  - smoothed interpolation possible (additional expense, accuracy?)



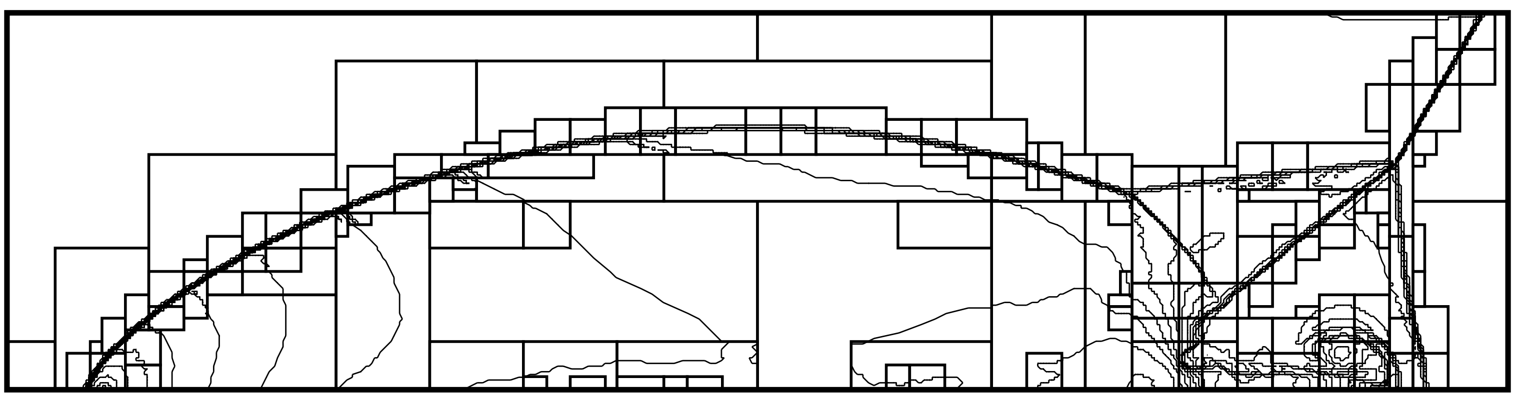
- May need to recompute quantities like pressure (incompressible flow), freestream preservation correction, etc. (re-initialization can be expensive)

Chombo is a collection of C++ libraries for implementing block-structured adaptive mesh refinement (AMR) finite difference calculations.

• Mixed-language model: C++ for higher-level data structures, Fortran for regular single-grid calculations.

• Reuseable components. Component design based on mapping of mathematical abstractions to classes.

• Build on public-domain standards: MPI, HDF5.



History

Chombo is an outgrowth of a single group developing AMR algorithms for a broad range of complex multiphysics applications: astrophysics, combustion, shock dynamics, porous media flows, interfacial dynamics, turbulence, ...

Previous / related work: BoxLib (CCSE /LBNL), KeLP (Baden et. al., UCSD), FIDIL.

The range of applications leads to a particular form of the standard design criteria for software:

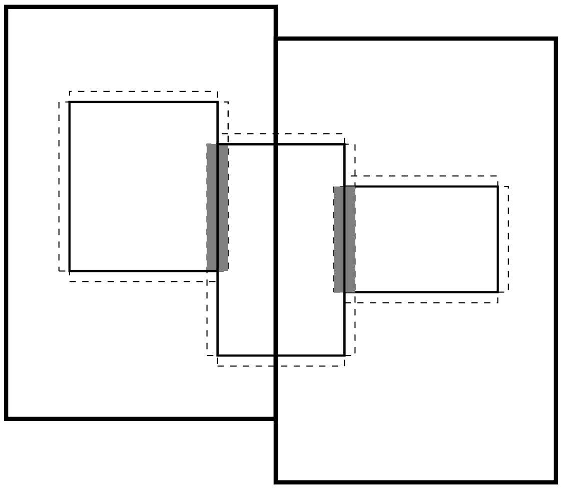
- Expressiveness: how well does the programming notation match the natural mathematical description of the algorithm.

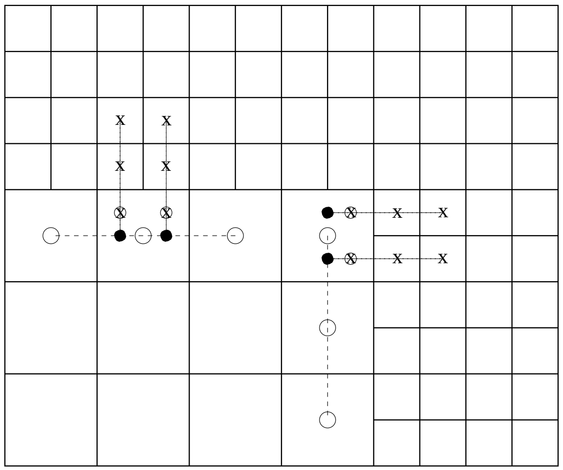
- Reuseability: how difficult is it to introduce new capabilities, or to apply the software to new problems.

- Performance: how difficult is the program to tune, and how well does the tuned code perform.

Expressiveness

C++ abstractions map to high-level mathemematical description of AMR algorithm components (Chombo is an AMR developer′s toolkit).





`BoxLayoutData, LevelData` classes: encapsulate data defined on collections of rectangles distributed over processors.

```
LevelData a; BoxLayoutData b;
a[Ind] = ...; // indexing returns reference to rectangular
array.
a.copyTo(b); // Copies valid data in a to b.
a.exchange(); // copies valid data in a to ghost cells in a.
```

Layered Design

The layers in Chombo correspond to different levels of functionality in the AMR algorithm space.

- Layer 1: Multidimensional arrays and set calculus, data on unions of rectangles mapped onto distributed memory.

- Layer 2: operators that couple different levels: conservative interpolation, averaging between AMR levels, interpolation of boundary conditions at coarse-fine interfaces, and refluxing operations to maintain conservation at coarse-fine interfaces.

- Layer 3: implementation of multilevel control structures: Berger-Oliger time stepping, AMR-multigrid iteration, Berger-Rigoutsos grid generation.

- Layer 4: complete adaptive PDE solvers. Current examples include elliptic, parabolic, hyperbolic equations, incompressible flow.

- Utilities : HDF5 - based parallel I/O, ChomboVis visualization tools based on VTk, Fortran support tools.

Reuseability

There are four mechanisms used in Chombo to enhance reuseability.

- Substitution of procedures that conform to an interface specification, e.g. substitution of different Fortran subroutines for integrating various hyperbolic systems of conservation laws on a single grid.

- Composition: higher-level functionalities can be obtained through composition of different combinations of lower-level components. For example, the Layer 2 tools for computing interlevel operations are used to implement a broad range of elliptic, parabolic, and hyperbolic AMR operators and solvers.

- Reuse of control structures across various data structures using inheritance. Berger-Oliger time-stepping requires only pointers to a pure virtual base class that defines what is meant by advancing the solution in time, computing the time step, etc. The derived class holds the data.

- Reuse of container classes using templates. `BoxLayoutData<T>`, `LevelData<T>` are templated on the multidimensional array type `T`. `T` can be array of reals, integers; binsorted collections of particles; arrays with sparse multivalued subsets.

Performance

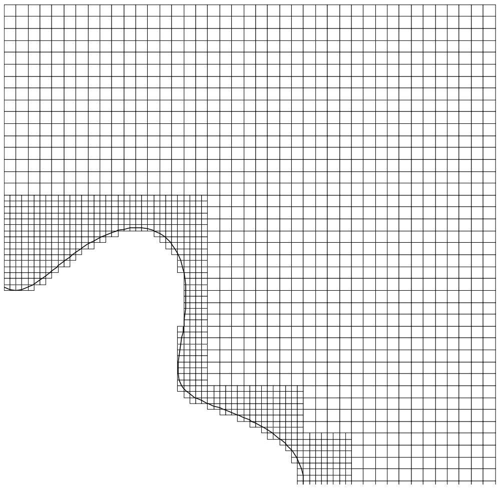
Serial Performance:
High performance is obtained by computing bulk-rectangular grid operations in calls to Fortran 77. The C++ rectangular array library provides access to pointers to the data stored contiguously in row-major order. Chombo includes a macro package to facilitate the use of this interface that also allows one to write dimension-independent FORTRAN. This allows one to limit the use of C++ array operations to implementing sparse irregular calculations, which leads to acceptable performance ($\geq$ 80% of CPU time spent in Fortran for gas dynamics).

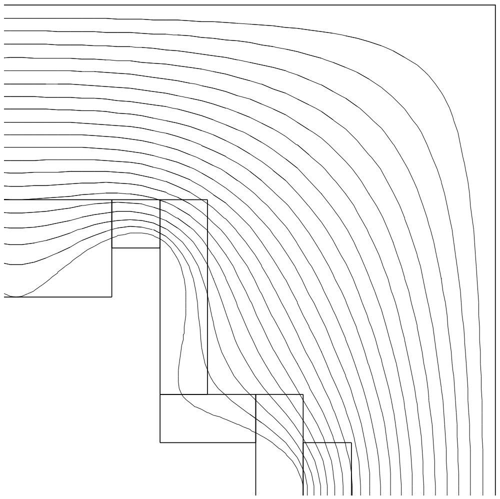
Parallel Performance:
For AMR, parallel performance is highly problem dependent. In applications where it has been required, algorithms have been shown to scale to 100's of processors (CCSE / BoxLib).

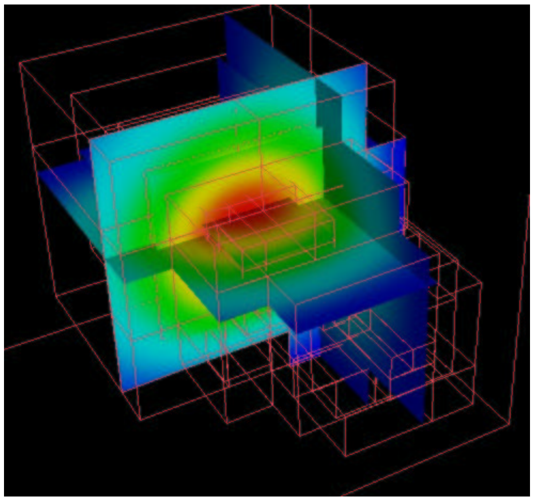
Availability

Chombo can be downloaded from the Berkeley Lab AMR website. The Applied Numerical Algorithms Group at LBNL has a long-term commitment to supporting Chombo, with major enhancements to its capabilities (Cartesian grid treatment of geometry, particle-grid methods) required to support the DOE SciDAC applications in accelerator modeling, magnetic fusion, and combustion.

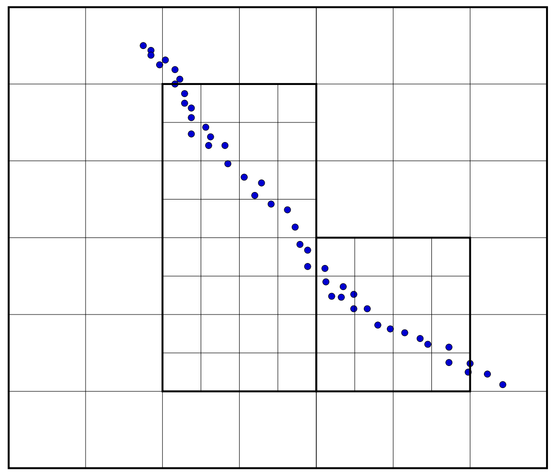
Stand-alone C interfaces to much of the Level 1 and level 4 functionality are also under development with the AMR/CCA forum.







Particles and AMR in Chombo



- Currently implementing PIC algorithms in Chombo
- Templated container classes have allowed straightforward extension of basic classes to particles.
- Algorithmic Issues:
  - Transfers of particles across coarse/fine interface boundaries
  - Particle $\rightarrow$ Grid and Grid $\rightarrow$ Particle transfers in the presence of refinement boundaries (modified stencils)
  - preventing self-induced effects.